10-708: Probabilistic Graphical Models 10-708, Spring 2014

5 : Sum-Product Message Passing

Lecturer: Eric P. Xing *Scribes: Alex Loewi, Zhe Zhang*

1 Recap of Elimination and Marginalization

This lecture begins with some slides from the previous Lecture 4, at slide 48.

1.1 Elimination Algorithm and Complexity

In the previous lecture, we discussed the algebraic variable elimination algorithm. This algorithm allowed us to do exact inference on conditional probabilities given some evidence. The process of this algorithm algebraically involved getting an elimination ordering, moving summation operators into the probability factorization as far 'inside' as possible, and then at each step creating intermediate terms by summing over a variable of non-interest.

We also saw that this process is represented graphically at each step by removing nodes from the graph and creating edges for intermediate elimination cliques. Each intermediate term (after summing over a variable) corresponds to an elimination clique at each step. It is a clique since all the variables in the clique have not been able to be eliminated yet.

The complexity bottleneck for the elimination algorithm is the largest such intermediate elimination clique in the process. In Slide 51, the example of the star and tree demonstrate this complexity. On the same graph, two different orders of elimination can lead to very different run times. For example in star, if you eliminate the edges first, you never have a clique with more than two variables. However if you start with the center, you have a clique of n-1 of the variables.

Example: Ising model – what's the inferential complexity? Let's start at a corner – eliminating down a row (or column) will result in a whole column-sized clique, with the clique adding an edge after each elimination in the row. Thus, the tree width is n, i.e. the size of one row (in the grid).

Transitioning to the Lecture 5 slides, a quick note on complexity as well. It is NP-hard to find the best elimination ordering.

1.2 Limitations of elimination

"Today we are going to explore the third benefit of having a graph," which is doing efficient inference. In elimination, the algebraic elimination corresponds exactly with graphical elimination. As mentioned, each intermediate step is related to an elimination clique. If we consider a tree of these cliques, then each elimination step can also be thought of as message passing on a clique tree (slide 54).

The key insight here is that messages can be reused, allowing you to pre-compute clique functions/factors, and get new marginals quickly. If we change our mind on the probability of interest, this is helpful, avoiding redoing work. Some things are intrinsically hard to infer, but many useful ones are tractable.

2 Inference on Trees

On Slide 6, we introduce the graphical concept of trees. The valuable property of a tree in this context is that there is ONE UNIQUE path between every pair of nodes. This point will come up in more detail later, where it will be shown that if a message is sent along two DIFFERENT paths, it can actually have two different values. For now, we will just consider directed and undirected trees, not poly trees.

Undirected and directed trees are equivalent. If you have edge and node potentials for everything (which are all you need because trees' largest cliques are edges), it's trivial to see how conditional probabilities can be represented as clique potentials, meaning the representation of directed and undirected (but not poly) trees can be formally identical. The math on Slide 7 displays this equivalence.

2.1 Example sum-product elimination on a tree

A natural ordering is to eliminate from the bottom leaves to the top. However, any query root can be chosen and treated as a directed graph with edges pointing away from it. $m_{ji}(x_i)$ is the factor resulting from eliminating variables from below up to i, which is a function of x_i.

On Slide 9, for an undirected graph (which holds generally), we define the sum-product equation for each message originating from a node, x_j to a node x_i. It depends on the node potential, ψ_j of node j as well as all the neighbors of x_j (excluding node x_i of course). Generally, node j takes the information (messages) from its neighbors and sums over them (including its own information), thus grouping them together into a message for node i. The product operator includes the previous information (which depends on x_j) and the sum operator then adds in relevant information (and also eliminates x_j).

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j) \right)$$

In the algorithm implementation, the key protocol is that a node can only pass a message along after it has received messages from its *other* neighbors.

2.2 Sum-product Complexity and Two-pass Algorithm

We're interested in the complexity of computing all the node marginals. Naively this complexity of message passing is O(NC) (where N=nodes, C=complexity of one complete passing/clique bottleneck). However, actually, you can get it in two passes: 2C, or O(C)! One pass is the forward pass and the other is the backward pass. Pass one message in each direction along each edge. This allows full calculation of all possible messages, two for each edge, as seen in Slide 16. Once these messages, we can calculate any marginal we are interested just through the sum-product equation.

2.2.1 Implementation

The sequential implementation involves choosing a root, collecting messages up to the root, and then going back down to distribute messages. There is also the parallel synchronous algorithm. In this, have the node wait until it has received messages from all but one nodes – then compute a message, and send it to the remaining node. You can actually just *continue* to send the messages until everything converges, and everything will be fine – this is an implementation detail.

3 More General Graphs

What about non-trees? To show the difficulty, let's work with a diamond graph (in order of 1,2,3,4), just to think about it. We'll just have nodal and edge potentials for simplicity. Let's say we want to deal with the marginal probability of x_1:

It's proportional to all elements that contain it as an argument. If we start with a message order for $1 \leftarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$

$$p(x_1) \propto \phi(x_1) m_{4\rightarrow 1} m_{2\rightarrow 1}$$

$$p(x_1) \propto \phi_1 \sum_{x_4} (\phi_4 \phi_{4,1} m_{3\rightarrow 4}) * \sum_{x_2} \phi_2 \phi_{2,1}$$

$$p(x_1) \propto \phi_1 \sum_{x_4} \phi_4 \phi_{4,1} * \sum_{x_3} \phi_3 \phi_{3,4} m_{2\rightarrow 3} * \sum_{x_2} \phi_2 \phi_{2,1}$$

After we expand $m_{2\rightarrow 3}$, this is the end, because there aren't any messages coming into x_2.

Now, change the message passing direction, from $1 \leftarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$. What you will find is that you get two different answers, when you pass messages two different ways. It starts the same way,

$$p(x_1) \propto \phi(x_1) m_{4\rightarrow 1} m_{2\rightarrow 1}$$

As you expand however, you see that the result for $p(x_1)$ will be different than with the different ordering of arrows. This time, node 4 has no messages going into it.

$$p(x_1) \propto \phi_1 \sum_{x_4} (\phi_4 \phi_{4,1}) * \sum_{x_2} (\phi_2 \phi_{2,1} * m_{3\rightarrow 2})$$

$$p(x_1) \propto \phi_1 \sum_{x_4} (\phi_4 \phi_{4,1}) * \sum_{x_2} \phi_2 \phi_{2,1} * \sum_{x_3} \phi_3 \phi_{2,3} m_{4\rightarrow 3}$$

Thus, belief propagation is only valid on trees.

4 Transforming non-trees

So, what do we do? Let's find a way to deal with non-trees. How about – we turn non-trees INTO trees! We do this by creating factors for every relationship in a graph. On slide 20, we see an example of this. Each factor involves all of the variables involved in each of the potentials, as demonstrated by the algebra.

Depending on how you factor, sometimes you don't actually improve anything. On slide 21, in example two you still get a ring, which is what you had to being with. However example 3 (which represents the same graph) is a STAR which is a tree, so you've actually won something.

4.1 Message Passing on Factor Trees

There are, however, some added attributes. Now we have different kinds of messages, since you have bipartite graphs with two types. We have to calculate the two kinds of messages differently between the two types of nodes.

As displayed on Slide 23, for a message going from a node to a factor, the node takes in messages from all other factors (μ type messages) and combines them to send. For a factor sending a message to a node, it

takes all messages from the other nodes (ν type messages) and combines them *with* its factor potential (which we got in the derivaton of the factor tree) to send. The algorithm protocol is similar, in that nodes/factors wait for information from its other neighbors before it can send. The marginal probability is denoted on Slide 24 and is similar to the sum-product.

The key insight here is that tree-like graphs can be easily turned into trees and allows belief propagation. This happens without loss of information as demonstrated earlier by the algebra in the transformation capturing the original potentials.

How does this help us for our earlier Ising-model example? About the best you could do with an Ising-model lattice is to turn the lattice into a chain, by making a factor for each column. This is bottlenecked by the largest factor.

4.2 MAP probability instead of marginal

Going over this quickly because it is straightforward, to get the maximum a posterior probability for some node given evidence, the operation is the same as the sum-product algorithm except you're just replacing SUM with MAX. The Jordan chapter does a clear and thorough breakdown of this replacement and also delves into how one also optimizes for optimal configurations, x^*.

5 The Junction Tree Algorithm

What if the graphical model is not a tree-like graph? This requires the junction-tree algorithm. It is actually pretty complicated (and obsolete) BUT – it demonstrates the principles of many graphical inference algorithms, so it's good for general understanding.

Let's build a tree where we can be guaranteed correctness. To do this, we return to the idea of clique trees we saw in the elimination algorithm, as reminded on slide 33. Each intermediate, elimination clique in the process of elimination captures the induced dependencies of each elimination.

First we need to moralize the graph since the parents are dependent. This gets these nodes into the same clique, so we need to connect these cliques. We start with the moral graph, and then build a clique tree on the moral graph. This still maintains the information from a BN, because of the v-structures.

To operationalize this, we need to create potentials on each of the neighboring cliques, as seen in Slide 35. There is one potential for each clique in the clique tree. This is referred to as a 'separator' potential. This is useful because you can directly make use of clique potentials to represent marginal distributions. We can now consider neighboring cliques that have some overlap of nodes S. There is a good demonstration of this transformation algebraically on slide 37.

Thus, once you have an estimation of the clique potential, you will be able to term the probabilistic inference queries into local operations, on the potential functions that contain the argument from the inference. Once we introduce evidence or some perturbation, we message pass in order to 'recalibrate' the tree so that the cliques still represent the correct cluster marginals of the variables they represent.

We want local consistency. Though, for overlapping S, there are multiple ways to obtain marginal information, since they can come from either side. We can marginalize over either piece they're connected to. To ensure consistency, these pieces must be the same, so we need to renormalize the forward-update and the backwards-update on the neighboring cliques. The Invariant Joint property on Slide 39 demonstrates the need for this consistency and demonstrates how we might normalize the messages from both sides to ensure this consistency. Thus, the augmented forward and backward updates respect this.

This is referred to as the Hugin update (and the previous sum-product update is the Shafer-Shenoy update).

5.1 Correctness of Junction Tree and Triangulation

Another issue arises – we have this message passing algorithm, but is it correct? The problem before was that we had two ways of passing messages, which didn't agree and lead to different results. So we turn things into a tree, and we're done. *But: Are all converted trees correct?* Again, the diamond graph we had trouble with earlier continues to be troublesome.

Maybe the make-tree function gives you back stuff you can't work with. For example, can you get a variable in two cliques that aren't connected? This is demonstrated in the clique tree in slide 41. The node C is not connected in the clique tree, so we may not get global consistency for C.

So – how do we ensure that something is consistent? In fact, there's nothing to ensure that we get this "junction-tree" property, also called the "running intersection" property or "triangulation" property. This is actually a problem. No guarantee they're correct. How do we AVOID this? Just don't create trees randomly! *Create* cliques if you don't have them (ADD edges, which is okay because it happens in moralization, for example, and these nodes are still related). You have to triangulate first by creating chord edges, THEN split things up into a clique-tree. This will avoid separated cliques that share a common node.

The resulting clique tree from the triangulated graph will have the running intersection property. If a node appears in two cliques, it appears everywhere on the path between the cliques, thus ensuring global consistency.

Once you've made a triangulated tree, *don't confuse these with a graphic model*: these are just representations. Edges are just places along which messages can be passed – they're not violating dependence properties.

So what's left: HOW do we triangulate? Triangulation is much more of an art. Remember that if we really knew, we would have resolved PvNP, so again, only heuristics exist. That said, the general sketch of the Junction Tree Algorithm is presented on slide 47, bringing together the steps we've discussed thus far (moralizing, triangulating, clique trees, and message passing).

In a final example, we see how message passing makes the HMM archetype example quite straightforward to solve with a forward and backward pass, and how they correspond to the elimination algorithm.